REMARKS

The above amendment is submitted in response to the Examiner's Action of 7 April 2005.

The Examiner's Action objected to the abstract of the disclosure as exceeding 150 words in length. The Applicant has amended the abstract of the disclosure with a new abstract of the disclosure as set forth herein [page 26 of this paper] consistent with the requirements of 35 U.S.C. §111 and MPEP § 608.01(b).

The Examiner's Action objected to Applicant's recitation and use of the trademark "Java™." Applicant has amended recitation of the mark "Java™" within the specification consistent with this objection [pages 5 – 6, 8 – 13, 15 – 18, and 20 – 25 of this paper]. Use of capitalization and/or generic trademark symbols or terminology is not possible or practical for references to ".java" in file extension and source code files.

By amendment as detailed herein, Applicant has corrected typographical errors in the specification at: (1) page 59, lines 16 – 19 [page 14 of this paper]; and (2) page 90, lines 13 – 17 [page 19 of this paper]. Neither of these corrections presents new matter to the specification.

Also by amendment as detailed herein, Applicant has provided a Statement Regarding Copyrighted Material [page 4 of this paper] and a new paragraph concerning the source code materials on CD as originally filed with this application [page 7 of this paper]. Neither of these paragraphs presents new matter to the specification.

The Examiner has requested Applicant's cooperation in clearly disclosing and claiming one invention. Applicant has cancelled original claims 1 – 57, and herein submits new claims 58 – 83 [pages 27 – 36 of this paper] consistent with the Examiner's request. As more fully detailed in this response and set forth below, the Applicant provides references to the specification for each new claim and distinguishes each new claim from the reference cited by the Examiner for rejection of the original claims.

The Examiner objected to claims 44 and 47 as being in improper form. Applicant has cancelled original claims 44 and 47.

The Examiner rejected original claims 1 – 57 as being directed to non-statutory subject matter under 35 U.S.C. §101. This rejection was predicated upon original claims 1 and 2 being directed to non-statutory subject matter. The Examiner further rejected original claims 3 -57 under 37 U.S.C. §101 for containing a listing of features without being interrelated and as not remedying the cited deficiencies of original claims 1 and 2. Applicant herein provides new claims 58 – 83 directed to computer-implemented and executable methods and systems performing useful acts as more further discussed herein below for each new claim.

The prior art cited by the Examiner is noted.

The specification of the present invention (hereinafter "Specification"), page 76, lines 18-20, discloses the principal novelty of the computer-implemented and executable methods and systems performing useful acts of the present invention, which are: (1) the "capture-pattern" of new claims 59 and 60; (2) the "do-pattern" of new claims 61 – 65; (3) the subjunctive expressions of new claims 66 – 68; and (4) the production rule template of claims 69 – 77. These methods provide computer-implemented forms of regular expressions to achieve a practical application, and which are novel in not only the grammatical form, but also the abilities they offer to the programmer, and in the way in which side-effects are integrated into all of the regular expression forms. New claims 78 – 83 disclose systems of implementation of these methods, also not anticipated in the art.

The presented new claims relate to computer-implemented and executable methods/systems for side-effects of regular expressions in general, and the "do-pattern", the "capture-pattern", the "subjunctive", and the **parameterizable** production rule grammar. Novelty for each new claim *in seriatim* is discussed below, based on a comparison with the FRIEDL reference as cited by the Examiner and with specific reference to the supporting text of the Specification of the present invention.

As an accommodation to, and acknowledgment of, the Examiner's request for particularity in disclosing and claiming one invention, and in the interests of expediency, the Applicant respectfully submits Specification references and, where appropriate, distinction over the FRIEDL reference cited by the Examiner in rejecting the claims as filed, as more particularly detailed on the new claims as follows.

- **New Claim 58**

This claim states the principal novelty and claim of the invention to be a "method of integrating side-effects into regular expressions". This is disclosed as a general objective of the Specification at page 3, lines 21-23. This is disclosed as an aspect of the "do-pattern" in the Specification at page 84, lines 5-6. The Specification at page 50, lines 12-18 further discloses that "capture-patterns" and parameterizable rules are modeled by low-level "do-patterns", thereby also involving the integration of side-effects into regular expressions. The "subjunctive" claims below demonstrate also that the side-effects of the primary are integrated into the subset construction convolution that models the subjunctive expression.

This claim also states how side-effects are integrated into regular expressions of the invention, and new claims 59-77 demonstrate that the methods of doing so are novel.

- **New Claim 59**

This claim discloses that the regular expression grammar of the invention includes a composition form that completely specifies: (1) the sub-composition being captured and which determines the (sub-)match to data; (2) the directed variable location of the capture; (3) the scope of the variable, into which capture is directed, to be any scope available at the same scope in which the sub-composition being matched/captured is defined.

There are two functions this method gives the programmer that are not anticipated in the art, the first being that the directed capture of data is directly to a named variable (of type String) available in the same scope as the "capture-pattern", the second being that this directed capture can occur at any composition or sub-composition level of a regular expression.

This disclosure is provided in the Specification at page 6, line 20 through page 7, line 4. The grammar is provided in section 8.3.5.4.2 of the Specification at page 53, line 19 through page 55, line 3. Examples are introduced in section 8.4.2.1 of the Specification at page 76, line 21 through page 83, line 19. In fact, most examples of the present invention as detailed in the Specification include directed capture to named variable references of type String. In terms of the ability of capture to occur at any sub-expression level, such as within repeats, this is stated in section 8.4.2.2 of the Specification at page 84, lines 19-20, and disclosed by example in the Specification at page 85 lines 4-5, and line 22.

In terms of cited reference, FRIEDL, page 137, lines 4-7, discloses that only Python and .NET offer named capture grammars, but that these grammars allow capture to a "named location" rather than a "named variable". A named location is not the same as a named variable, a defect that manifests itself in trying to write a single expression that captures multi-dimensional data, explored further in new claim 66. FRIEDL, at page 41, line 1 teaches the nature of capture variables in Perl and "most modern regex engines" in which capture variables refer to text matched by parenthesized sub-expressions and are referred to in code **outside** of the regular expressions, **after** the match has been successfully completed. This is quite different from a design pattern identified by the present invention, in which capture is directly to either a temporary variable scoped to a DoPattern, or also to an "out" param or "in out" param of a production rule. FRIEDL, at page 43, line 2, has a section heading entitled "Intertwined Regular Expressions" in which the problem caused by moving the results of capture from variables such as $1 into code that uses them means that in the art, the work of capturing is partially inside the regular expressions but is then completed outside the regular expressions, a difficulty that the FRIEDL reference, at page 48, line 4, calls "intertwined". In contrast, the "capture-pattern" of the present invention accomplishes capture directly to the variable of choice, entirely within a regular expression composition.

In terms of "mimicking named capture" in Perl, FRIEDL, at page 344, line 6 through page 346, teaches an approach for mimicking this aspect of the invention, as well as shortcomings of such approaches inherent in the Perl engine. FRIEDL, at page 345, line 6, states: "Finally, it's my hope that Perl will eventually add named capture natively..."

- **New Claim 60**

The Specification, at page 78, lines 23-28, discloses new claim 60, a generalization of new claim 59, in that the "capture-pattern" not only allows capture to named variables but also to any valid (in-scope) reference of type String, such as a String-array indexing expression. The Specification, at page 79, line 5, and at page 85, line 27, discloses and illustrates examples of this grammar form in action.

FRIEDL discloses neither anticipation of new claim 59 (see above) nor its generalization which is new claim 60.

- **New Claim 61**

Section 8.3.5.4.1 of the Specification, at page 50, line 22 through page 51, line 14, discloses all of the elements of new claim 61. These elements describe the semantic (meaning and function) and syntactical aspects of the "do-pattern". The Specification, at page 51, lines 10-11, further discloses that the meaning of the "do-pattern" is to "attempt to match the embedded sub-expression..." Thus the "do-pattern" is a "positive assertion" (to use the terminology of FRIEDL), meaning that it matches characters rather than positions.

FRIEDL, at page 327, line 5, teaches an ability to execute "embedded code" within a regular expression sub-composition, ascribed only to Perl. However, as disclosed in the Perl documentation, this Perl form is "zero-width" meaning that it matches a position rather than characters of data. Grammatically, the Perl form is similar to a special case of the "do-pattern" as follows: do (...; null; ...).

The syntactic choice of the invention was made principally as a way to establish a scope (for defining temporary variables), such scope being available not only to the embedded code (pre-list and post-list of statements) but also the matchable sub-expression (and any of its "do-patterns"). The zero-width assertion form of Perl does not grammatically lend itself to defining scope, nor is such attributed to Perl as discussed under new claim 63.

The sematic effect of the "do-pattern" is dissimilar to the semantic effect of the Perl "embedded code" assertion, as seen by comparing the reductionist "do-pattern" mentioned above – do (...; null; ...). The semantic difference is the "if and only if" guarantee concerning whether or not the code-statements will execute, which is explored under new claim 64. The practical success of the "if and only if" guarantee is demonstrated by the working examples of the invention, but fundamental semantic difference is claimed under new claim 64 rather than superiority. The question of superiority is answered ultimately by users of the invention.

- **New Claim 62**

The Specification, at page 18, lines 2-10, discloses that the pre-list and post-list statements of the "do-pattern" can use any of the statement grammars of the invention. This includes defining variables scoped to the "do-pattern", as discussed under new claim 63. FRIEDL, pages 295-299, teaches the concept of "dynamic scope", and states in particular at page 296, line 3, that "the call to *local* does not create a new variable..." This discussion and the heading "Using local in an Embedded-Code Construct," FRIEDL, at page 335, together show that the "embedded code" construct does not allow the programmer to define variables **scoped** to a regular expression. The "do-pattern" of the present invention **<u>does</u>** allow a variable-defining statement in its statement lists, such variables being scoped to the "do-pattern."

- **New Claim 63**

The Specification at page, 7, lines 1-4, page 54, line 3-5, page 54, line 9, and page 106, lines

3-6, discloses that the "do-pattern" establishes a scope into which variables can be defined. The Specification further discloses and illustrates by examples where such scoped variables are used (after definition in the pre-list) in both the matching sub-expression of the "do-pattern" (Specification, at page 85, line 22, page 89, lines 11-16, page 106, line 20("temp"), page 106, line 21("pos"), page 106, line 23("pos"), page 121, line 14, and page 121, line 27) **and** in the post-list of the "do-pattern" (Specification, at page 85, line 25, page 89, line 18, and page 106, line 21("temp")).

FRIEDL, at pages 295-299, teaches that Perl has a "dynamic scope", and in particular (page 296, line 3) states: "the call to *local* does not create a new variable. local is an action not a declaration..." FRIEDL shows in this section that the dynamic scope concept relates to "global variables" being given temporary values, and then restored if backtracking occurs.

- **New Claim 64**

The important elements of this claim are that (1) side-effects of a "do-pattern/"capture-pattern" are modeled as instructions, and that said instructions execute **if and only** the pattern is involved in a winning match, (2) that the "do-pattern" translates its pre-list and post-list to instructions, and (3) that the "capture-pattern's" instructions are implied by modeling the "capture-pattern" as a low-level "do-pattern". The Specification, at page 6, lines 13-16, page 6, lines 20-24, page 53, lines 14-17, and page 84, lines 11-15, discloses (1), the **if and only if** aspect. The Specification, at page, 51, lines 10-27, page 53, lines 14-17, page 84, lines 11-13, and page 138, lines 5-11, discloses (2), that side-effects of such compositions are translated to instructions. The Specification, at page 6, lines 16-19, page 50, lines 17-20, page 53, lines 8-11, page 54, line 29 through page 55, line 3, and page 123, lines 1-7, discloses (3), that the "capture-pattern" is modeled as a low-level "do-pattern".

Relative to the art, only Perl has a concept of "embedded-code". FRIEDL, at page 327, line 5, teaches that Perl's embedded code statements execute each time "it's reached during the application of a regex..." This statement and Perl's documentation show that embedded-code is executed each time a match is attempted, whereas the present invention's side-effects (low-level "do-pattern" statements) execute if and only if the corresponding match expression is part of the "winning" match (Specification at page 13, lines 4-7, page 53, lines 14-17, and page 84, lines 11-16). FRIEDL, at page 336, line 2 further discloses the problem with variable updates in embedded-code, that is, that "the code executed via the embedded-code construct is not somehow 'unexecuted' when its part of the regular expressions is 'unmatched' via backtracking." This statement shows that the embedded-code construct of Perl is executed "on-the-fly", as the NFA engine encounters that regex "position", and that when backtracking occurs, the code cannot be "unexecuted". FRIEDL, at page 336, lines 4-6, teaches a work-around for "variables", allowing the backtracking engine to "undo" a variable change, but this "undoing" can only be accomplished with global-variables demarked "local" (the dynamic scope feature), whereas not all embedded code can be undone. The present invention does not have this as an issue since side-effect instructions are not executed on the fly, but are truly executed as side-effects once the "winning" match has been determined.

- **New Claim 65**

The Specification, at page 169, line 1 through page 171, line 22, discloses that a "winning" match must not only determine the characters (and number thereof) matched by automata execution, but also which set of side-effect instructions are associated to the "winning" thread. The Specification, at page 169, line 1 through page 171, line 22, further discloses that the "winning" set of side-effect instructions must correspond to ambiguity-resolving rules in the presence of side-effects. These ambiguity-resolving rules are disclosed throughout section 8.5.2.2 and its sub-sections, and relate to the presence of side-effects in the context of union compositions, concatenation compositions, repeat compositions, and iterate compositions.

FRIEDL, at pages 174-175, and 177-182, also discloses the importance of understanding the match semantics of each regex engine, and discusses how such can differ from that defined by the POSIX standard. This claim governs the selection of at most one set of side-effects to execute for the entire regular expression composition against data. When multiple automata paths exist that can match the same characters of data, the only way to support the **if and only if** claim of new claim 64 is to determine a "winning thread" based on a comprehensive set of ambiguity-resolving rules (knowable and controllable by the programmer), and to only execute the instruction-arcs (as side-effects) traversed by the "winning thread." The claim is therefore novel to the extent that no other engine in the art permits the equivalent of "do-patterns" that include the **if and only if** guarantee of new claim 64.

- **New Claim 66**

The semantics of the subjunctive convolution are explained in the Specification, at page 59, line 4 through page 60, line 2, page 90, line 2 through page 91, line 9, and page 222, line 10 through page 223, line 1. Section 8.4.2.3 gives examples of both kinds of subjunctives, showing their power and semantics in action.

The use of the primary to qualify the secondary is disclosed in the Specification, at page 93, lines 30-33, and page 99, lines 5-10. The definition of the subjunctive is given in the Specification, at page 5, line 24 through page 6, line 5, in which it is stated that the secondary is used to "restrict" the matches of the primary. The Specification, at page, 91, lines 6-9, also discloses the concept in simple terms: the secondary "prunes" the set of possible matches of the primary. The fact that the side-effects of the secondary term are ignored/inhibited is disclosed at Specification, at page 59, lines 16-19. The fact that the primary term contributes side-effects as well as matching is disclosed in the Specification, at page 59, line 20 through page 60, line 2.

The many examples of the Specification, at page 89, line 29 through page 100, line 29, using the subjunctive explore its value to the programmer in achieving increased match specificity (Specification, at page 91, lines10-13). The historical reasons for the name *butnot* are explored in the Specification, at page 238, lines 5-14, showing that butnot is like "set difference", but that simple "set difference" via a "minus" operator would not convey clearly the primary/secondary relationship of the two terms, in which the side-effects implied by only the primary should be allowed to execute. The choice of names for both forms of the subjunctive as relates to "set arithmetic" of "intersection" and "subtraction" is also disclosed in the Specification, at page 222, line 20 through page 223, line 1.

FRIEDL, pages 59-67, teaches the existence in Perl and related art of "lookahead/lookbehind assertions". These Perl-like assertions would be semantically conveyed by keywords such as *followedby*, *notfollowedby*, *precededby*, and *notprecededby*, but are not offered in the invention. FRIEDL does not disclose any other types of "lookaround", and in particular does **not** disclose a "look-at-the-same-time" assertion. The subjunctive of the invention is same-time arrival qualification, which separates it from Perl-like engines and their "lookahead/lookbehind" assertions. Whereas a "lookahead/lookbehind assertion" of Perl-like engines is a position-pinned "zero-width" assertion, the subjunctive of the invention is a postive-assertion, meaning that the secondary must match/not-match the same data matched by the primary at the same time.

In terms of any related art offering set-operations such as "intersection" and "difference", FRIEDL, at page123, line 3, discloses only the Java™ regex package, and its set-operations for character-classes. Character-classes match exactly one character, whereas the subjunctive of the invention allows arbitrary-sized matches based on whatever the primary can match, as restricted by the secondary. This feature of the Java™ engine is a special case of the subjunctive.

In terms of "intersection" and "difference" set operations (possibly offered outside of the related art considered by FRIEDL), none are offered as a composition of two general regular expressions, and none allow a primary term to retain its side-effect characteristics.

In terms of this claim being a method of new claim 58, the subjunctive grammar form has a primary, and permits, encourages, and handles the embedding of side-effects into the primary term via "do-patterns" and "capture-patterns".

- **New Claim 67**

The Specification, at page 59, lines 10-12, discloses that for the *but* grammar (relative to the match set of the primary) "any matches which the secondary does not also accept are discarded". The Specification, at page 206, lines 8-10, uses the phrase "same-time arrival" to describe the implementation of the subjunctive construction's accept condition. The Specification, at page 222, lines 10-25, discloses the semantics of the subjunctive in terms of the "accept condition" being to match/not-match at the same time. The semantics of the *but* are disclosed in the Specification, at page 222, lines17-18, and show that any candidate accept states for the primary are discarded if the secondary does not "accept" the data at the same time (in at least one way). The Specification, at page 59, line 20 through page 60, line 2, discloses that the secondary in no way changes the side-effects implied by the primary's match.

A detailed hypothetical/slow execution of the subjunctive is given in the Specification, at page 90 line 2 through page 91, line 9, to illustrate exactly the matching and side-effect behavior to be expected for the *but* form of the subjunctive. This hypothetical engine finds not only all possible matches that the primary can make to the data, but also the instruction-arcs encountered during traversal of such matches (Specification, at page 90, lines 7-12). This hypothetical engine also finds all possible matches that the secondary can make (Specification, at page 90, lines14-17). The word "subjunctive" on line 13 would be more easily understood as "secondary", in contrast to the preceding lines. The pruning step for the *but* grammar is given in the

Specification, at page 90, line 24 through page 91, line 2, which is that matches from the primary that do not have a corresponding match in the secondary are pruned. That is, the secondary must be able to match the same data at the same time, or else the potential match of the primary is pruned. The "same time" concept is given in the Specification, at page 90, lines18-21.

The fact that the actual engine (which is much faster) has the same semantic effect is disclosed in section 8.5.4.

- **New Claim 68**

The Specification, at page 59, lines13-15, discloses that for the *butnot* (relative to the match set of the primary) "any matches which the secondary also accepts are discarded". The Specification at page 206, lines 8-10, uses the phrase "same-time arrival" to describe the implementation of the subjunctive construction's accept condition. The Specification, at page 222, lines10-25, discloses the semantics of the subjunctive in terms of the "accept condition" being to match/not-match at the same time. The semantics of the *butnot* are disclosed in Specification, at page 222, lines19-20, and show that any candidate accept states for the primary are discarded if the secondary "accepts" the data at the same time. The Specification, at page 59, line 20 through page 60, line 2, discloses that the secondary in no way changes the side-effects implied by the primary's match.

A detailed hypothetical/slow execution of the subjunctive is given in Specification at page 90, line 2 through page 91, line 9, to illustrate exactly the matching and side-effect behavior to be expected for the *but* form of the subjunctive. This hypothetical engine finds not only all possible matches that the primary can make to the data, but also the instruction-arcs encountered during traversal of such matches (Specification, at page 90, lines7-12). This hypothetical engine also finds all possible matches that the secondary can make (Specification, at page 90, lines14-17). The word "subjunctive" on line 13 would be more easily understood as "secondary", in contrast to the preceding lines. The pruning step for the *butnot* grammar is given in the Specification, at page 91, lines3-5, which is that matches from the primary that **do** have a corresponding match in the secondary are pruned. That is, the secondary must **not** in any way be able to match the same data at the same time, or else the potential match of the primary is pruned. The "same time" concept is given in the Specification, at page 90, lines18-21.

The fact that the actual engine (which is much faster) has the same semantic effect is disclosed in section 8.5.4.

- **New Claim 69**

The Specification, at page 19, lines 21-24, discloses the reusability of regular expressions as encapsulated production rules with parameters. The grammar form of the invention's production rule (including a discussion of the syntax and usage of its parameters) is specified in section 8.3.5.4.3. In particular, the Specification, at page 56, lines 1-6, discloses that the body of the production rule is defined to be any valid regular expression composition, and the Specification, at page 56, lines 7-24, discloses how such "body" of rule may use the rule's parameters. The enhanced re-usability made possible by parameters of a production rule is explored in various sub-claims below.

FRIEDL, at page 303, line 3 through page 306, teaches only one tool in related art that offers named, re-usable encapsulations of regular expressions, calling them "regex objects". FRIEDL, at page 303, line 3, teaches a use-case summary that reads: "Regex objects are used primarily to encapsulate a regular expressions into a unit that can be used to build larger expressions, and for efficiency..." FRIEDL **cannot and does** not disclose that Perl's encapsulations are parameterizable, because they are not. The regex "encapsulations" of the invention are called production rules rather than "regex objects" **and** they offer the **novelty** of parameterizability.

In terms of related art, which includes "regex engine tools", such as .NET, Java™, Python, Tcl, etc., all discussed in FRIEDL in great detail, none of the tools aside from Perl offer the equivalent of encapsulated "regex objects", and again, Perl's "regex objects" are **not** parameterizable.

- **New Claim 70**

This claim is a spring-board for new claims 71 and 72, and is supported and further discussed below in those new claims for each of the two "effects" of parameters, those that parameterize "side-effects" and those that parameterize "matching" or "eating behavior".

- **New Claim 71**

The Specification, at page 56, lines14-18, discloses that "in", "out", and "in out" parameters may be used within pre-list and post-list statements of a production rule (thereby parameterizing side-effects of the rule) and that "out" and "in out" parameters may be used within "capture-patterns" (thereby also parameterizing side-effects of the rule). The Specification, at page 105, lines19-22, page 105, lines 27-30, and page 106, lines16-25, discloses an example of an "in" parameter being used to parameterize side-effects of a rule, via both "do-patterns" and "capture-patterns". The Specification, at page 107, lines 22-23, page 108, lines 8-11, further discloses an example of an "out" parameter being used to parameterize side-effects of a rule, via "capture-patterns"... The Specification, at page 111, lines1-2, and page 111, lines14-17, discloses an example of an "in out" parameter being used to parameterize side-effects of a rule, via a "do-pattern".

FRIEDL does not teach or otherwise disclose any prior art in which encapsulated "regex objects" have any kinds of parameters at all.

- **New Claim 72**

The Specification, at page 56, lines7-13, discloses that "in" params can be used to build the production rule's "eating"/recognition characteristics. The Specification at page 103, line 23 through page104, line7, likewise discloses anticipated use cases of the "in" variety of parameter being used to control matching characteristics of the production rule. The Specification, at page 104, lines 8-11, further discloses another example of the "in" parameter to a rule being used to parameterize the matching characteristic.

FRIEDL does not teach or otherwise disclose any prior art in which encapsulated "regex objects" have any kinds of parameters at all.

- **New Claim 73**

The Specification, at page 114, line1 through page 115, line 2, discloses that a instantiated rule **with** side-effects can be passed as the target parameter of another rule. The Specification, at page 115, lines 2-4, discloses that the side-effects of the bound target are also executed too, so long as they are part of the winning thread's traversal. In this example being discussed, those side-effects execute as part of the target being "found" by the "find-nth" rule. The side-effects of the target rule are disclosed in the Specification at page 115, lines7-9, corresponding to the line of code in the Specification at page 116, line 4, which has a "do-pattern". The successful triggering of the side-effects of the target is disclosed in the Specification, at page 115, lines 19-21. The target is instantiated in [INVENTION, 117/2]. The target is passed to the "find-nth" rule in the Specification, at page 117, line 4.

FRIEDL does not teach or otherwise disclose any prior art in which encapsulated "regex objects" have any kinds of parameters at all.

- **New Claim 74**

The Specification, at page 110, lines 26-32, discloses the concept of "binding" an actual parameter value to the instantiation object, and why this is important. The Specification, at page 148, lines 1-20, further discloses how this is done. Throughout the Specification, the term "instantiation" is used in conjunction with production rules, to indicate that they are properly to be thought of as "bound" to parameters. The Specification, at page 55, lines 5-7, discloses that the "instantiation of a production rule produces an actual pattern (regular) expression object." The various examples in section 8.4.2.4 demonstrate that the bound and instantiated production rule is type compatible with the *Pattern* datatype.

FRIEDL discloses no related art that has a similar concept.

- **New Claim 75**

The Specification, at page 117, lines 30-122, discloses an example that demonstrates the step-by-step composition of a single production rule which performs the complete job of capture of multi-dimensional data. The Specification, at page 121, lines 20-30, shows the code for such a completed composition. This production rule not only performs the complete capture of multi-dimensional data, but it directs its capture to a parameter data structure of the rule. Throughout the code of this example, "do-patterns" are used (lines 32/33/45/46 of the example's code) as well as "capture-patterns" (line 16 of the example's code), all within the bodies of production rules.

Perl does not have the concept of parameterizable rules, **so it cannot encapsulate the complete job of directed capture such that the destination of the capture is an argument/parameter of a production rule**. That is, Perl's encapsulation of regular expressions, called "regex objects" is fully disclosed in FRIEDL, at pages 303-306, and does not have the option of parameters.

- **New Claim 76**

The Specification, at page 101, line 24 through page 102, line2, and page 102, lines 25-30, discloses an example of the reject literal being used with the conditional operator as a base case for an invalid selector. The Specification, at page 116, line 12, discloses a second example of the reject literal being used for the base case along with the conditional operator, within a production rule body.

FRIEDL discloses no related art that has a similar concept.

- **New Claim 77**

The Specification, at page 62, lines 2-4, discloses the semantic effect of the *inhibit* grammar, as having the same matching effect of its sub-composition, but stripping out all of its side-effect producing instruction-arcs. The Specification, at page 115, lines 4-6, page 115, lines 7-9, page 115, lines 16-18, and page116, line 15, discloses an example of *inhibit* being used within a production rule, because the target parameter p must be matched N times, where "p" and "N" are parameters, and where "p", when actually passed could contain side-effects, as shown in the example, and wherein those side-effects must be "inhibited" the first N-1 matches.

FRIEDL discloses no related art that has a similar concept.

- **New Claims 78 and 79**

Section 8.5 discloses the implementation of the virtual machine (8.5.1), the automata composition engine (8.5.2), the subset construction techniques (8.5.3 and 8.5.4) and the automata runtime execution method (8.5.5). The fact that method 1 of the invention is made possible in the form of a programming language, virtual machine (FORTH-like), and regex engine, is discussed early and throughout the Specification.

In terms of the implementation of the regex engine of the Specification, the principal terms describing novelty, used and described in detail in various sections, are "instruction-arc", "arcnum-sequence", "winning thread", and "pruning". The fact that all of these concepts contribute to the step-by-step runtime automata execution algorithm **without character re-examination** is disclosed by step 8 of the Specification at page 234, line 23 through page 235, line 4.

The Specification does not re-examine characters. Rather, it extends the DFA-notion of maintaining a state of all possible matches, to a thread-set of all possible matches by all possible instruction traversals of a "winning thread". Its novelty is principally supported by the "arcnum-sequences", which are not found elsewhere in the art, and its "pruning step" (based on those arcnum-sequences), which is also not found elsewhere in the art.

- **New Claim 80**

The terms "side-effect", "instruction-arcs", "do-patterns", "capture-patterns", "arcnum-sequences" are used throughout the Specification. The way in which instruction-arcs imply the existence of arcnum-sequences (both being put into the composition graph) is discussed in section 8.5.2 and its subsections. The automata execution technique, including the step of automata-thread preference (pruning, step 9) is disclosed in section 8.5.5.

This claim is stating, in simple terms, the discovery of the invention, that instruction-arcs create a potential source of ambiguity, and that potentially multiple traversal paths of such instruction-arcs exist for "accepting" the same data, and that a preference must be made. This preference is makable based on arcnum-sequences, which reflect the ambiguity-resolving rules associated with unions, concatenations, repeats, and iterates. This preference is made aggressively in the automata runtime engine, further explored in claim 25.

The concept of "arcnum-sequences" to determine a "winning thread" is not found in the reference. FRIEDL, at pages 157-169, discloses a much different approach to preference in the backtracking NFA engines of the art, such as Perl.

- **New Claim 81**

The subset-construction technique in the presence of "arcnum-sequences" and "instruction-arcs" is described in section 8.5.3 and shown to correctly find the "winning" match, according to the ambiguity-resolving rules of section 8.5.2.2. Since only the "winning" match is needed, the subset constructed graph is suitable for execution. The "winning" match is based on automata thread preference, based in turn on arcnum-sequences, and results in a set of instruction-arcs to be executed as the proper side-effects of the match.

The use of arcnum-sequences to establish thread-preference in the presence of general side-effects (such as from "do-patterns") is not found in the art. Nor is a "winning" set of side-effects guaranteed in the art. For example, claim 7 shows that Perl executes embedded-code on-the-fly.

FRIEDL does not teach or otherwise disclose anticipation of a subset construction technique that includes the proper handling of arcnum-sequences and instruction-arcs, because no related art has the concept of arcnum-sequences.

- **New Claim 82**

The subset-construction technique that performs the subjunctive-convolution is described in section 8.5.4.

The subset-construction technique to perform a subjunctive-convolution is not found in the art, because such a construction is not elsewhere attempted in the presence of "instruction-arcs" and "arcnum-sequences".

- **New Claim 83**

The "pruning step" of the automata execution method is step 9 of section 8.5.5.

This "pruning step" is not found in the art or FRIEDL reference because it is predicated on the existence of "arcnum-sequences" also not found in the art or FRIEDL reference.

CONCLUSION

With the filing of this response and amendment, the application and all claims should be in condition for reconsideration and allowance. For all the reasons advanced above, the Applicant respectfully submits that the application is in condition for reconsideration and allowance and that action is earnestly solicited.

Dated: July 6, 2005.

Respectfully submitted,
/Charles L. Thoeming/
Charles L. Thoeming
Registration No. 43,951

BIELEN, LAMPE & THOEMING, P.A.
1390 Willow Pass Road
Suite 1020
Concord, CA 94520
(925)288.9720
(925)288.9731 Facsimile

Certificate of Mailing by Express Mail

I hereby certify that this correspondence and all related papers are being sent by Express Mail No. ED 514126945 US the USPTO under correspondence placed in a pre-paid envelope addressed to: **MAIL STOP AMENDMENT**, Commissioner for Patents, P. O. Box 1450, Alexandria, VA 22313-1450, on July 6, 2005.

/Charles L. Thoeming/
Charles L. Thoeming, Registered Representative of Applicants
Registration No. 43,951

July 6, 2005
Date of Signature